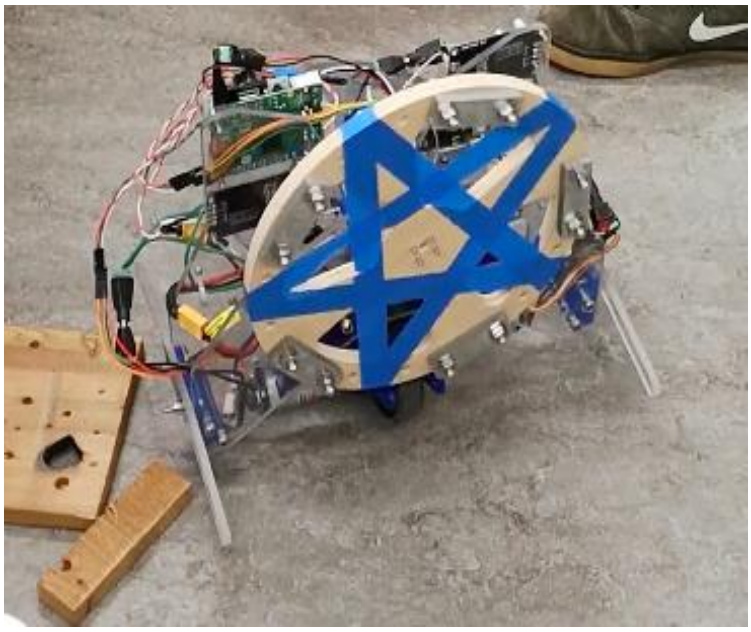
Aiur
AI Unicycle Robot



Curtis Huebner
Anmol Jawandha
Koosha Rezaiezadeh
Cheng Xie

Project Sponsor:
Dr. Michiel Van de Panne

Project 1868-1869
Engineering Physics 479
Engineering Physics Project Laboratory
The University of British Columbia
January 7th, 2019

Executive Summary

A physical platform for enabling reinforcement learning (RL) research was designed and developed for the project sponsor, Michiel Van de Panne. In consultation with the sponsor, it was identified that a self-balancing unicycle robot, along with an overhead camera to track it, would make for an effective platform. This is because there is prior work demonstrating that it is possible to build a self-balancing unicycle and get it to move from point to point. Furthermore, RL algorithms can theoretically let a unicycle do more than what is possible with classical control methods.

In addition to the hardware of the robot, supporting software was developed. This includes a physics simulation of the robot, control and computer vision software for point-to-point navigation, CAD files of the robot, and a low-level API for the control of the robot.

The design process involved several steps. The robot was built by first developing basic simulations to make sure that when constructed, the robot could physically balance. From there, a test platform was developed to validate the control mechanism. Afterward, the full robot was constructed, and its ability to balance along each axis was tested. Finally, the full platform for point-to-point navigation was built by adding the overhead camera and writing a computer vision software to track the robot. The robot's ability to balance and move from point-to-point was then demonstrated.

Two mechanical issues were identified that we recommend resolving to better facilitate reinforcement learning. First, the robot has difficulty turning in place due to an unreliable servo and simplistic control algorithms. Second, the hub connecting the motor shaft to the drive wheel comes loose after extended use. Before using the platform for RL, a mechanism should be developed to reset the robot automatically when it falls, and the hub and servo should be upgraded.

Table of Contents

Introduction

Background

Reinforcement Learning (RL) is an emerging branch of Machine Learning theory that models an *agent* interacting in an *environment*. This general-purpose formalism, combined with the power of deep learning, has given rise to powerful algorithms (popularly referred to as *Deep RL* algorithms). Recent successes of these algorithms include mastering the games of Go, Chess, Atari, DOTA2. In addition to mastering these games, these algorithms have also shown success in the field of robotics on tasks such as object grasping, object manipulation, and even locomotion [1].

Dr. Michiel Van De Panne, our sponsor, is a professor at UBC with interests in graphics, physics-based locomotion, and reinforcement learning. Some of his recent work aims to control robots for locomotion via reinforcement learning [2]. However, *learning* optimal control on real robots is hugely time-inefficient. One way to sidestep the difficulties of learning in the real world is to perform learning in a simulation of the real robot and its environment. These controllers, once learned in simulation, can then be transferred to the real robot in several ways. Some work in this area of "Sim to Real" [7] has allowed robots to successfully use simulation controllers to learn object manipulation and block-stacking in the real world [3]. Thus, learning control on real robots is an exciting area of research for our sponsor.

Unicycle control is an example of a real-world control problem. There are previous groups that have put together self-balancing unicycles that are kept upright with reaction wheels [4]. Although the aforementioned groups all use classical control methods to balance their unicycles, it's also possible to control them with RL. Finally, away from the equilibrium of standing upright, unicycles display interesting nonlinear dynamics, and are capable of interesting actions, such as counter-steering and leaning into turns [5].

Problem Statement

The overarching objective of our project is to design and fabricate a robot along with the software to interact with it. These will serve as a research platform for our sponsor to test out RL algorithms for real-world robotic control. To enable easy implementation of custom RL tasks and

algorithms, this platform should provide a general API for executing actions that actuate the robot and receive observations gathered from sensors that describe the entire state of the robot. In consultation with our sponsor, we chose a unicycle robot as the hardware platform, due to the mechanical simplicity and potential for complex dynamics as described earlier. This choice naturally admits RL tasks such as maintaining in-place balance and point-to-point navigation.

Scope

To address the stated problem, our project encompasses the following objectives:

1. Design and construct a unicycle robot system that is physically capable of maintaining self-balance. This includes designing actuators that have sufficient power and response speed, sensors that are capable of perceiving the full state of the robot at a sufficient frequency as well as a microcomputer capable of executing the desired action.
2. Demonstrate the physical capability of our robot to perform interesting tasks that could be suitable for RL by using traditional control algorithms to achieve in-place self-balancing and point-to-point navigation.
3. Define and implement an API for the hardware functions of the robot. This includes access to the sensor readings as well as actuator outputs. As a robotic research platform, this would allow easy implementation of custom control algorithms on the robot.
4. Implement an example of remote control of the robot from a program running on a computer for the task of point-to-point navigation. This is a desirable quality for many RL algorithms that can be resource intensive and require access to libraries that may not be easily set up on the microcomputer.
5. Create a simulation of our robot closely matching the real robot that permits testing of control and learning algorithms as well as enabling exploration of “Sim to Real” transfer of RL policies.

Our primary goal is the development of a robotic research platform as defined by the first three items listed in the scope above. Originally, an additional stretch goal of our project was to use RL algorithms to learn faster point-to-point navigation. However, this was not pursued in this project due to the time constraints.

Organization of the Report

The report is organized into a discussion of the unicycle design and the design process. This is followed by a summary of the results, the conclusions of the project, and a list of deliverables. Finally, we present recommendations for the sponsor.

Discussion

This section of the report will first go over the final design of the robot, highlighting the mechanical, electrical and software components. We then discuss the design process, followed by critical design decisions and present intermediate tests that were done along the development process. We then conclude by going over the results of the project and offering recommendations for future developments. We also include important takeaways throughout our discussion.

Final Design

Figure 1 shows the final design of the unicycle robot, with labeled components. The robot is designed so it can move forwards and backward, lean sideways, and rotate in-place.

As depicted in the figure below, the final design of the robot uses two tightly pressed polycarbonate plates to hold the components. The DC motors of the bottom wheel are connected directly to 3D printed mounts that carry the main load of the robot. The electronics are mounted directly on the polycarbonate plates for easier accessibility. These polycarbonate plates also have mounting holes to allow for the connection of the training wheels and the colour markers. The main drive wheel, which is used to move the robot forward and backward, is controlled by two mechanically-coupled brushed DC motors. The sideways balance of the robot is achieved by a reaction wheel that is actuated by another brushed DC motor. Finally, turning on the spot is achieved with a horizontal reaction wheel controlled by a continuous rotation servo.

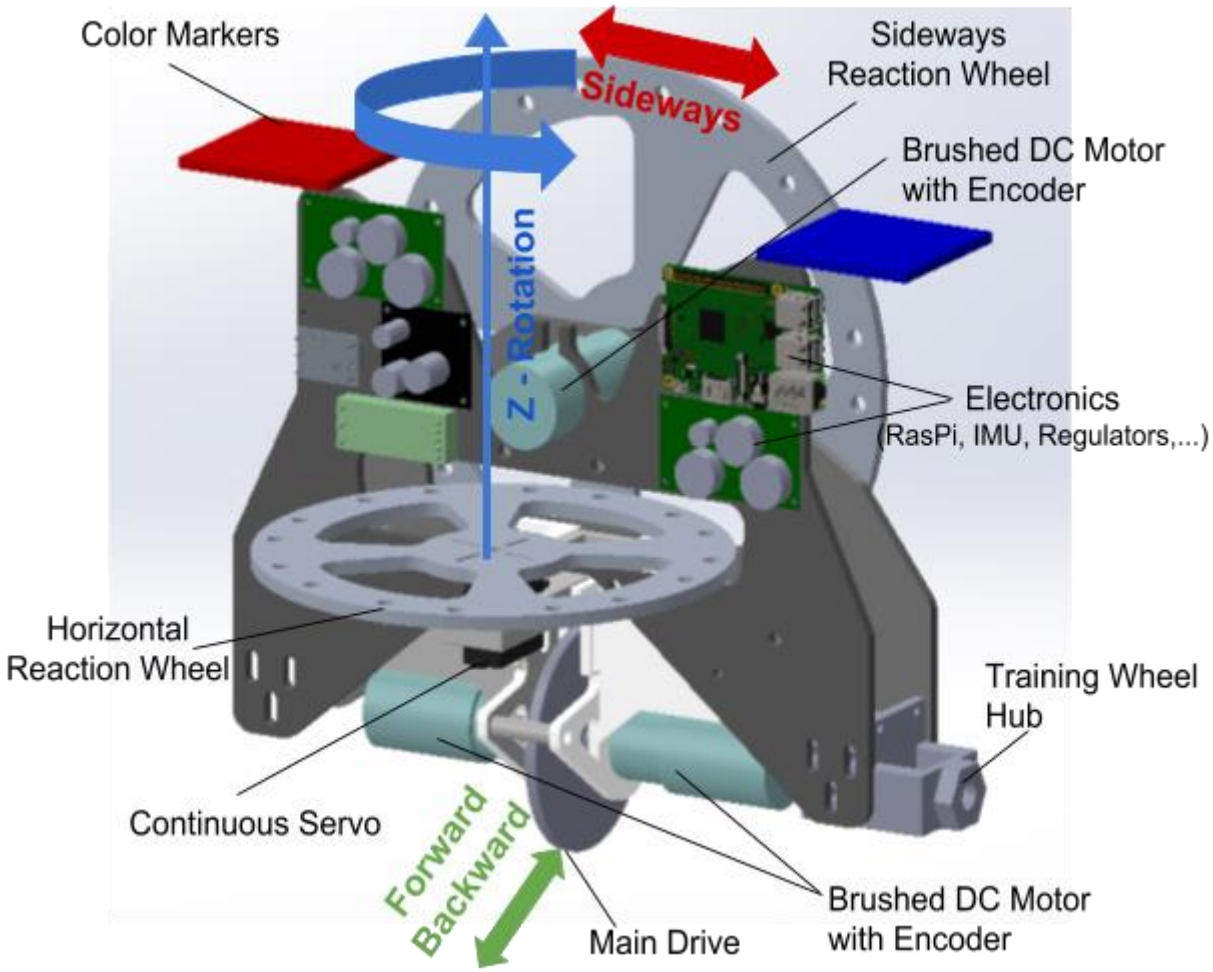


Figure 1. The Final CAD Design of the Unicycle, the major components are labelled in the picture. The picture also depicts the three main axes of movement.

It is desirable to have the robot's center of mass directly above the wheel's point of contact with the ground when the robot is vertical. For this reason, we make the position of the horizontal reaction wheel adjustable; adjusting this position changes the position of the COM of the robot.

The robot is controlled by a Raspberry Pi and receives data from a 6-DoF IMU. The power comes from two Li-Po batteries, providing 12 volts each. They can keep the robot running for about 10 minutes. The bottom wheel is directly driven by the battery, while the reaction wheels are powered by voltage regulators. The main wheel motors are powered directly by the unregulated batteries. Since the voltage of the batteries decrease as they deplete, the applied torque will lower as well. To compensate, we correct for this in software by measuring the battery voltage through an ADC and proportionally adjusting the PWM signal sent to the main wheel motors. See Appendix C for circuit diagrams.

The controller is divided into two levels. The low-level feedback controller runs on the Raspberry Pi. It maintains balance by actuating the motors in response to the local rotation and angular velocity of the robot. These are measured by the IMU and the speed of the motors derived from the wheel encoders. It can additionally admit control parameters that enable the high-level controller to modulate its behaviour. The high-level controller runs on a separate computer connected to an overhead webcam and measures the global 2D position and orientation of the robot through simple colour tracking as outlined in (Appendix D). The webcam is mounted approximately 2 meters overhead, which gives a 2.5 by 2.5 meter area for the robot to be detected in. Based on the 2D position, the high-level controller can then compute which control parameters to send to the low-level controller to accomplish a higher-level goal such as point-to-point navigation. The communication is facilitated by a wireless UDP connection (see Figure 2 for a system-level diagram). A more detailed explanation of the control algorithm is included below.

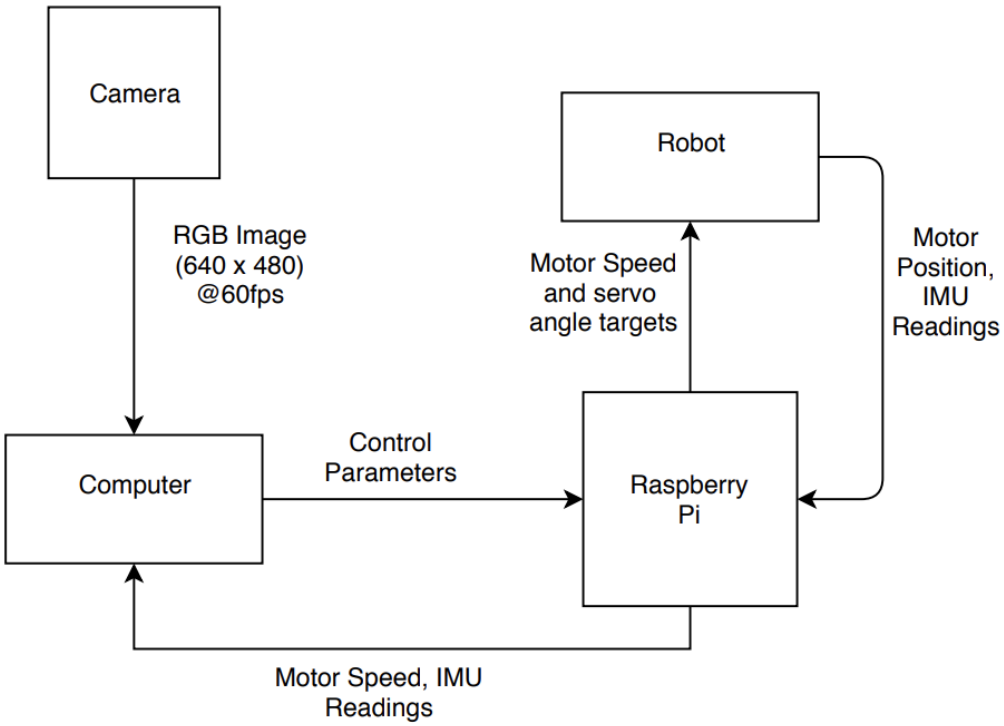


Figure 2. System-level overview of the controller. The computer receives feedback from the camera and the robot and provides the Raspberry Pi with the high-level control parameters. The communication between the Raspberry Pi and the computer happens wirelessly via Wi-Fi while the camera uses the USB interface. On the other hand, the balancing and the low-level control happens locally on the robot

Controller

One of the main objectives was to demonstrate the robot's capability through achieving point to point navigation in the overhead camera's field of vision. The controller used to achieve 2D point-to-point movement consists of 3 independent PD controllers, each controlling the robot along a different axis (the axes are shown in Figure 1). These controllers are named: sideways controller, forward-backward controller, and Z-axis controller. We detail the theory underlying these PD controllers and the corresponding body dynamics below.

Sideways control

The goal of the reaction wheel is to balance the robot sideways. To achieve this, we access the robot's IMUs to get sideways lean angle $\theta_{sideways}$, sideways lean velocity $\dot{\theta}_{sideways}$, and access the motor encoder to read the speed of the flywheel $\dot{\theta}_{flywheel}$.

To achieve balance, we apply the following PD torque:

$$T = I_{sideways}\alpha = p_{sideways}(\theta_{sideways} - \theta_{offset}) - d_{sideways1}\dot{\theta}_{fb} - d_{sideways2}\dot{\theta}_{flywheel}$$

Where α is the angular acceleration, $p_{sideways}$ is the proportional gain, and $d_{sideways1}$ and $d_{sideways2}$ are derivative gains. Note that a positive feedback gain is used on the speed of the reaction wheel. One potentially undesirable effect of this is that when the robot is held in place by hand, the positive feedback gain causes the reaction wheel to accelerate to maximum speed.

However, we cannot apply this torque directly. For a motor rotating at speed at $\dot{\theta}_{drive}$, the torque output is given by the following relationships:

$$T_{motor} \propto Current_{motor} \propto V_{PWM} - V_{emf}$$

where:

$$V_{emf} = k_{const.} \times \dot{\theta}_{sideways}$$

Please note that the output of the Raspberry Pi is the PWM signals and not the torque. Thus to apply the desired PD-torque, we apply the PWM signal

$$V_{PWM} = T_{motor} + V_{emf}$$

(See Appendix B for how back-emf constant $k_{const.}$ is calculated) A similar PD controller is used for forwards-balance, discussed in the section below.

A useful takeaway for achieving control using an IMU is that the gyroscope provides a very accurate measurement of the angular speed and is far superior to measuring angular speed through finite difference of the estimated angular position.

Forward-Backwards Controller

The goal in forward-backward control is to be able to move to and maintain any position in a straight line in the direction that the robot is facing.

We require a similar torque as applied in the case of sideways balance (discussed in the section above):

$$T = I_{fb}\alpha = p_{fb}(\theta_{fb} - \theta_{setpoint} - \theta_{control}) - d_{fb1}\dot{\theta}_{fb} - d_{fb2}\dot{\theta}_{drive}$$

However, in this case, the user specifies a point towards which the robot must move in the overhead camera's view. Given this point, we compute $\theta_{control}$. This parameter is an offset to the equilibrium position of the robot in the forward-backward direction and is proportional to the distance between the robot and its specified target. This offset parameter then forces the robot to lean in the direction of the target and ultimately move in that direction until it reaches the specified point.

In-place turning controller:

In place turning is achieved through the z-axis reaction wheel. We assume that a particular turning angle can be achieved by applying an instantaneous torque to the z-axis reaction wheel proportional to the size of the angle. Since we can only set the speed of the servo, we indirectly apply a torque to the reaction wheel through a sudden step in the desired speed. The size of this step function is proportional to the desired torque and hence also the size of the turning angle.

After turning, the angular motion of the servo needs to be decayed to zero in order to turn further without hitting the maximum speed limit of the servo. We accomplish this through a linear decay function that reduces the speed from the maximum speed to zero in approximately a second. By smoothly slowing down the reaction wheel over time, the instantaneous torque is lowered such that it does not overcome the static friction of the main wheel with the ground in the z-axis preventing the robot from rotating in the reverse direction. We observed that even at

this slow decay rate, some reverse rotation is still visible but doesn't significantly slow down the overall turn.

Combining controllers for point-to-point movement:

To achieve movement between two points in the camera's visual frame, we simply concatenate the in-place rotation control with the linear motion control, while keeping the sideways controller always running. We first turn to face the target while maintaining the current position. Then the linear motion forward-backward control is used to reach the target in a straight line. However, turning towards the target point is not very stable and is a limitation of the current system, as discussed in the section above.

Design Process

A self-balancing unicycle can have various sizes and shapes. As designers, our decision regarding the rough size of the robot was made mainly based on the overall budget (~$1.5k) and availability of parts and materials.

The process of the mechanical design of the unicycle was an iterative one. The design of the robot started with optimizing the performance metrics that would increase the stability and the controllability of the system. The most important of these performance metrics are minimizing the angular moment of inertias and the height of the center of the mass while increasing the torque and maximum speed of the motors. In later iterations, other factors such as ease of assembly and accessibility of the electronics parts were added to the performance metrics. For each design idea, we specified motors and servos that would provide a safety factor of ~2 in providing the required torque. In addition, we looked into various configurations for the placement of these parts. The configuration that best met the design requirements was then modeled in SolidWorks. We then exported the 3D CAD models to the PyBullet simulations where various linear and non-linear control algorithms were tested to check the stability of the system (See Appendix A for simulation results). The mechanical details of the CAD models were also discussed with the project lab. After incorporating feedback from the project lab, the whole design process was reiterated.

The safety factor turned out to be critical as when the robot was built, its total mass turned out to be 75% larger than expected. A low safety factor would have led to huge setbacks, as many components would have to be re-specified and reordered. It is very likely that without the safety factor the robot would not have been able to balance within the time constraints of the course.

Critical Design Decisions

Originally, our design for the sideways balance relied on two position-controlled arms actuated by two servos. The benefit of this would have been control over the side-to-side position of the center of mass of the robot. However, we found that the servos for the arm control presented numerous challenges. They were noisy, did not admit direct torque control, and did not provide any sensors to measure the position and speed of the arms. As a result, for control applications, we found off the shelf servos to be inadequate. Subsequently, we replaced the arms with a flywheel powered by a motor with built-in wheel encoder. In hindsight, it would have been better to start with the easier-to-control and previously validated flywheel design first, before attempting to control a novel arm-based design. See Figure 3 for a summary of the design iterations.

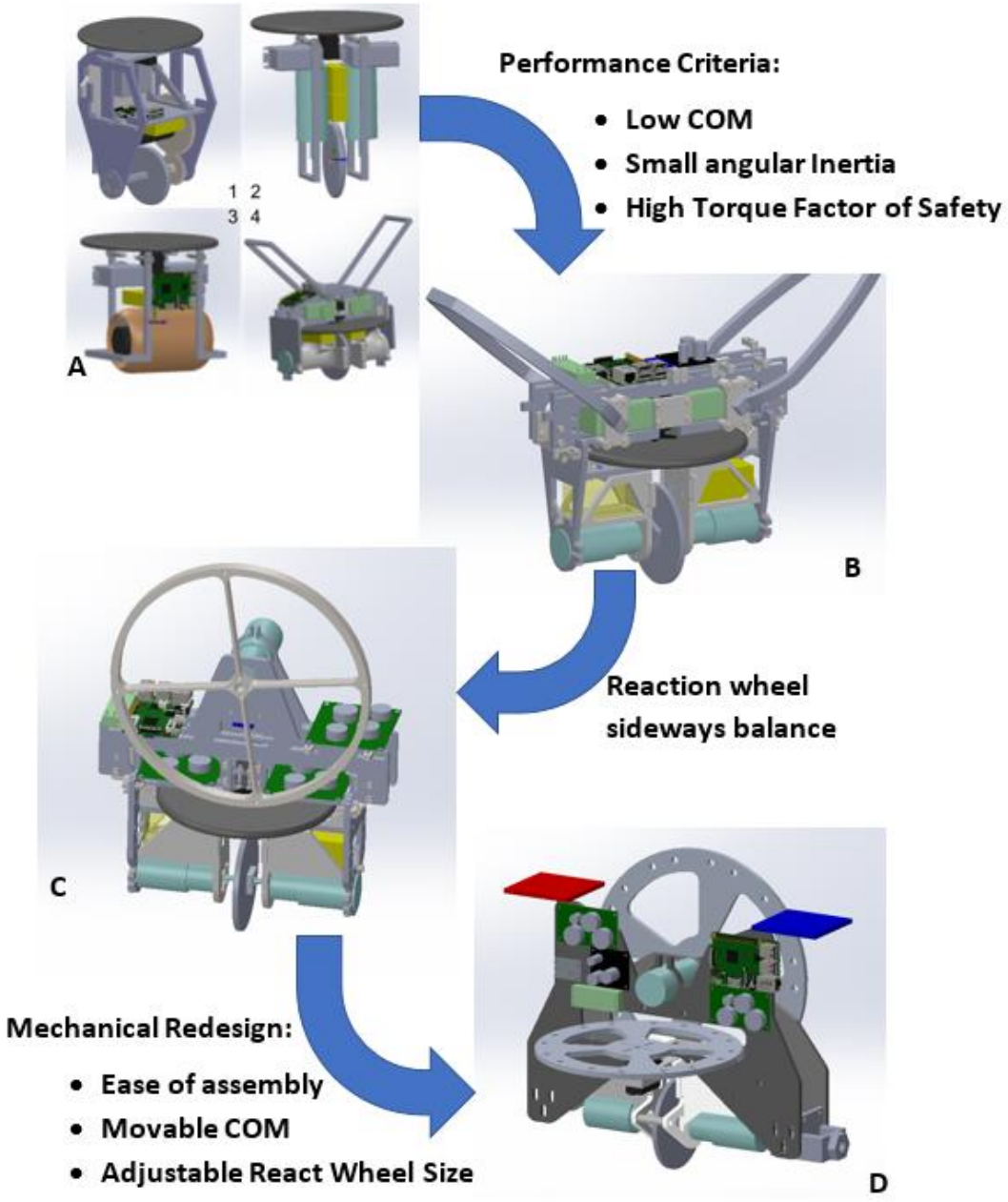


Figure 3. A Summary of the Mechanical Design Iteration. Multiple configurations (shown in figure 3.A) were investigated. According to the performance criteria, the details of the most optimum design was completed(Figure 3.B). Controller complexity pushed us to use a reaction wheel design for the sideways balance of the robot(shown in Figure 3.C). The robot was redesign to allow for ease of assembly, a movable COM, and the ability to change the reaction wheel size and mass (Figure 3.D) (See Appendix G for more details)

Intermediate Tests

The project was broken down into multiple phases. Each phase was concluded by testing a subsystem of our robot. Before constructing the robot, a test rig was built to validate that it was possible to balance the robot side-to-side with a reaction wheel (see Figure A9).

Development of the robot started with ensuring that sensors and actuators were working correctly. An objective of the project was to construct the robot with sensors that are capable of perceiving the full state of the robot at sufficient accuracy and frequency. We tested the delay time of the IMU with the test rig. By plotting the angle measured by the IMU as the test rig collided with an object equipped with a switch and comparing the discontinuity in the signal against the timing of the signal from the switch, we were able to roughly determine the delay in the IMU response to be 2ms. To measure the accuracy of the sensor, we equipped the test rig with a marker and tracked it using computer vision. While actuating the reaction wheel randomly, we record the IMU and the marker tracked angle simultaneously. We found that the recorded angles matched quite well even under large torque actuation except during hard collisions. This is sufficient; as we do not expect the robot to experience hard collisions during standard operation (see Figure A10).

Another objective was to ensure that motors were behaving as expected. To do this, PWM outputs were plotted against the rotation speed of the motor (see Figure A5). Doing this made it possible to test the motor actuators end-to-end, revealed software issues with the PWM signals output by the controller, and made it possible to determine the motor constant of the robot.

The physical capabilities of the robot were evaluated qualitatively by in-place balancing in the forward-backward and side-to-side axes separately. To do this, training wheels were employed to restrain movement in the other axis. These training scenarios are shown in Figure 4. The robot was able to balance indefinitely in both axes and was robust to small disturbances.

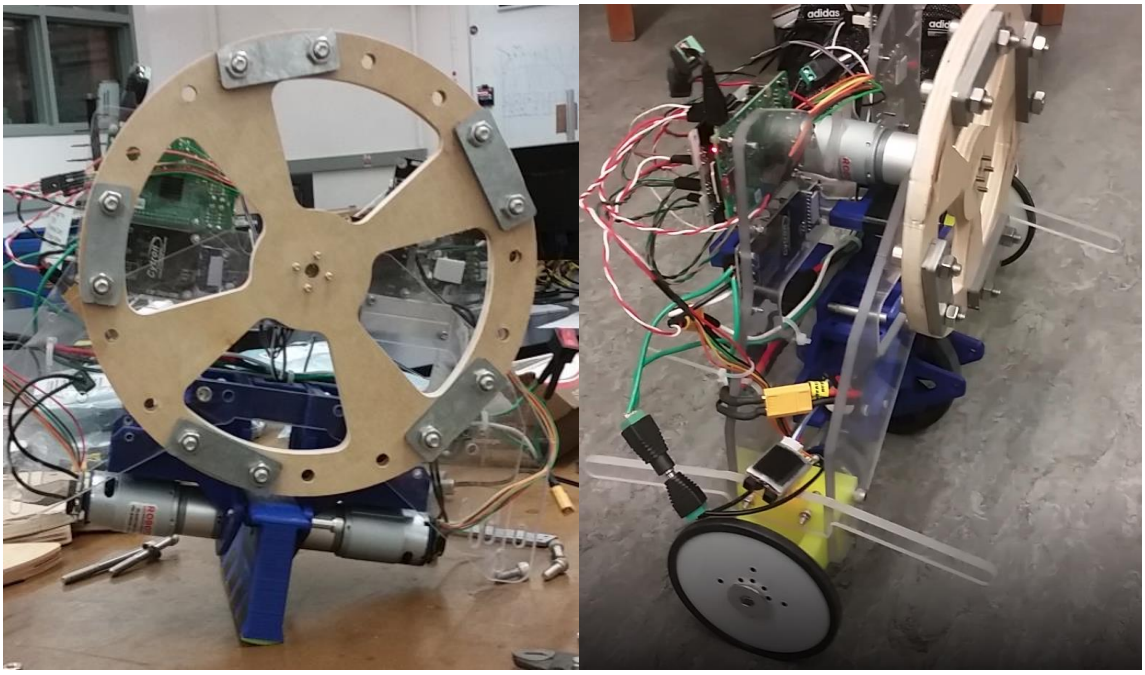


Figure 4. Single Axis Validation of the System. To isolate the motion to only the sideways balance, the wheel of the robot is replaced with a 3D printed hinge that has the same curvature as the wheel around the forward-backward direction and no curvature on the other axis (shown in the picture on the left side). On the right, the robot is fitted with training wheels on the side to isolate the movement to only the forward-backward direction.

In order to demonstrate that the robot is capable of full self-balancing, we remove the training wheels and combine the outputs of the individual controllers. We found that the coupled control works well and balances indefinitely when the robot is initiated from close to the equilibrium position.

Results

We designed and built a fully functional unicycle-based robotic research platform as defined in the scope of this document. We designed a point-to-point navigation task and implementation of a classical controller capable of achieving success on this task in order to demonstrate the full capabilities of the robot, all its subsystems, and performance with respect to the various criteria that were outlined in the scope of this report.

The point-to-point navigation task requires the robot to navigate to any point in the visual field of the webcam. The target can be chosen through clicking in the video stream of the GUI program

running on the remote computer. The classical controller communicates between the computer and the robot and achieves the task by first rotating the Z-wheel to correct its orientation and linear control to navigate to the destination. Although the robot is capable of moving from one point to another most of the time, it lacks reliable rotation about the z-axis and can only get to around 20 cm of the final destination. Occasionally, the robot will oscillate violently in the forward-backward direction following a turn. This is most likely due to the coupling effects in the dynamics of the separate axes. This would require more sophisticated algorithm to correct. Additionally, the motion is very slow with the robot capable of rotating about 10 degrees per second. This is due to insufficient torque generated by the continuous rotation servo used for controlling the z-axis reaction wheel.

An important objective was to be able to measure the full state of the robot through sensors, as most work in RL assumes access to the full state. All state variables relevant to the dynamics are accounted for, with the exception of the speed of the z-axis rotation wheel. It uses a servo and so does not report its current speed. We currently circumvent this problem by operating the z-axis rotation wheel at a low frequency thereby allowing it to come to equilibrium at which point the speed will match the desired target speed.

To enable easy implementation of custom control algorithms as required for RL research on the platform, we created a general-purpose python API for the robot hardware. The details of the API have been included in (Appendix F). This provides full access to the robot sensors and actuators. All of our control algorithms make use and demonstrate the functionality of this API.

A desirable quality for easy integration with existing RL frameworks is to allow remote control of the robot from a remote computer. The communication interface for this was implemented and demonstrated through the point-to-point navigation task whereby the computer operates at a lower frequency, sending control actions in the form of polar offsets from the target.

Conclusions

In this project, we were able to construct the hardware and software for an autonomous robotic unicycle that can act as a platform for RL research. We demonstrated the physical capability and robustness of the robot by achieving self-balance as well as achieving point-to-point navigation. We demonstrated the potential for the robot to enable experimentation for RL algorithms by developing software that simplifies interaction with the robot.

Although the robot is capable of moving from one point to another most of the time, it lacks reliable rotation about the z-axis. This could most likely be alleviated by improving the control algorithm further, taking into account a more detailed model of friction.

As a result, our project has achieved the primary objectives set out in our scope to enable further RL research by our sponsor, though improvements can be made. As such, we identify some key steps to be taken to further our overarching goal and detail them in the recommendations below.

Recommendations

We will first outline outstanding risks including the lack of observability of the full state, reliability of the z-axis rotation for point-to-point navigation and mechanical durability of the wheel hub. Then we will offer further recommendations for next steps to take in using our robotic platform for research in RL.

Some outstanding risks still need to be addressed. Although the full dynamical state of our robot is observable through our sensors, the angular speed of the Z-axis servo cannot be measured. Furthermore, we discovered that the torque it is able to generate is lower than we had anticipated, making rotation slow (10 degrees per second). These issues could be addressed by replacing the continuous rotation servo with a motor equipped with a wheel encoder as is used for the other wheels.

Although the robot is capable of roughly moving from one point to another, it lacks reliable rotation about the z-axis. Sometimes the robot will exhibit violent oscillations after completing a turn. We believe that this is mainly a limitation of the decoupled linear control algorithm, which

does not account for coupling effects between axes in the dynamics. More advanced control algorithms should be able to deal with these effects, and it provides an interesting challenge to explore with RL.

Furthermore, due to the lack of mechanical durability the dynamics of the robot change over time. This can mainly be attributed to the wheel hub. The set-screw connecting the motor-axle to the drive wheel must be re-tightened after continued use. This is most likely due to the aluminum wheel hub which shears under the high torques when holding the steel set screw against the steel D-shaft. We recommend purchasing a more durable wheel hub to replace the ones on the main drive wheel.

Through our software interface, we enable the user to implement any custom RL task. For getting started with the hardware API we recommend creating an RL task by extending the point-to-point navigation task. The missing components to achieve this are to define a suitable reward function to be optimized, and a reset mechanism to restart a trial.
The simplest reward function that is suitable to the task may be the negative total time taken to navigate to the target within an epsilon displacement with a final velocity under some epsilon value. So the longer it takes to arrive at the destination the lower the reward is. This is an example of sparse rewards and could provide a useful challenge for measuring the performance of exploration strategies in RL such as Hindsight Experience Replay [6]. The second obstacle is the reset function. The purpose of the reset function is to detect when the robot has failed the task and reinitialize it in a default state. We have already implemented fall detection for the robot and verified it through its use for automatic stopping. As for resetting to the initial state of the robot, it should start near equilibrium balance. We designed the robot safety guards and torque capabilities of the motors such that the robot could in principle right itself from a fallen position. Due to time constraints, we did not attempt to implement an algorithm to do so. Instead, all experiments were reset by manually placing the robot near its equilibrium position.

One motivation for our project is enabling research in “Sim To Real” RL. Although we have developed a fully functional dynamic simulation of the robot with realistic mechanical parameters and tested control algorithms with it, we have not quantitatively evaluated the accuracy of the simulation with respect to the real world dynamics of the robot. An interesting direction in this area for further research would involve automated testing between the

trajectories from real-world observed dynamics and those predicted by the simulation for different initial conditions and actions.

Deliverables

The main goal of the project was to design and construct an autonomous unicycle robot that can be used as a physical platform for RL research. The deliverables of this project will be submitted to our sponsor, Dr. Van de Panne, in the Department of Computer Science, UBC who has funded and provided the main motivation for this project. These deliverables include:

1. **A fully constructed unicycle robot:** The robot is programmed to be capable of communicating with a PC, balancing, rotating on the spot and moving in the forward and backward directions.
2. **The CAD design of the robot:** The sponsor will be provided with the full CAD models of the robot which will be useful for further future developments on the system, maintenance, and remanufacturing of any damaged components.
3. **A fully controllable model of the robot in the physics simulator along with a baseline controller:** This includes access to the URDF models of the robot and an interface to its sensors and actuators as well as the code for setting up the physical environment and controlling the robot around the simulated environment. This would allow the sponsor to test and verify the feasibility of various RL algorithms before moving to the physical world.
4. **Computer vision based software for 2D position control and high-level RL control of the system:** The system uses a camera that gets mounted above the testing field and uses computer vision algorithms to provide the robot with feedback regarding its current position and the target. This interface can be used for achieving point to point movement control for demonstrative purposes. It can also be used as a platform to test and verify more high-level RL algorithms such as object avoidance, path planning, etc.
5. **A low-level communication protocol for low-level control of the robot:** A low-level communication API between the robot and the computer is developed to allow for direct manipulation of all sensors and actuator on the robot. This system can be particularly useful for investigating more complicated RL algorithms that directly manipulate the robot on a lower level.

Appendices

A. Simulation Results

X-Axis Rotation Balance with the Arm Mechanism

This simulation tests the mechanical capability of the robot to maintain a sideways balance using the arm mechanism. Although we did not pursue the arm mechanism in the final design, this may be valuable information for attempting to add arms to our current design or construct a unicycle robot with arms in the future. The key parameters for the simulation and the selected values for our initial design are:

- Arm length 20cm
- Arm mass 200g
- Body COM height 12cm
- Body mass (no arms) 2.5kg
- Servo max torque 2.0Nm
- Servo max speed 1.2rps
- Height of arm joints 20cm

Some key assumptions of the simulation are:

- Servo is modeled as a DC motor with a linear speed/torque curve.
- All of the arm mass is concentrated near the end of the arm.
- Wheel contact is modeled by a perfect hinge joint to the ground assuming no slip.
- Mass distribution for the body is modeled as a uniform density cuboid.

The scenario that is simulated initializes the robot body at a certain angle offset from vertical at 0 velocity. The arms are assumed to be in their default location at 45 degrees from vertical and also have no velocity. Maximum possible torque is then applied to both servo motors to investigate the maximum offset we can recover from. A diagram of the geometry is included.

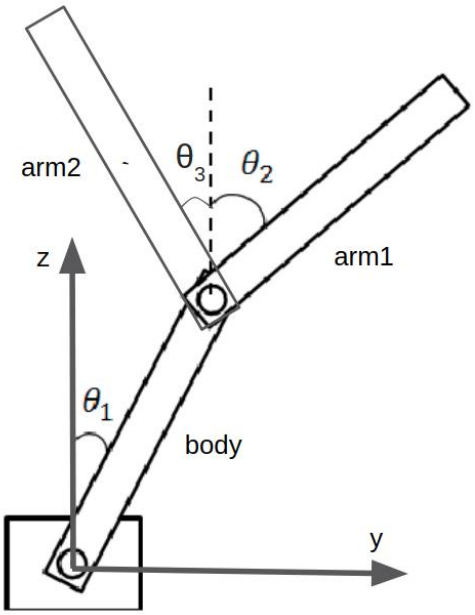


Figure A1. Diagram depicting the geometry of the x-axis simulation. All connections between bodies are perfect hinge joints. Angles are measured as offsets from vertical.

With the default values of our current design as outlined above, the robot is able to recover from a 10 degree offset.

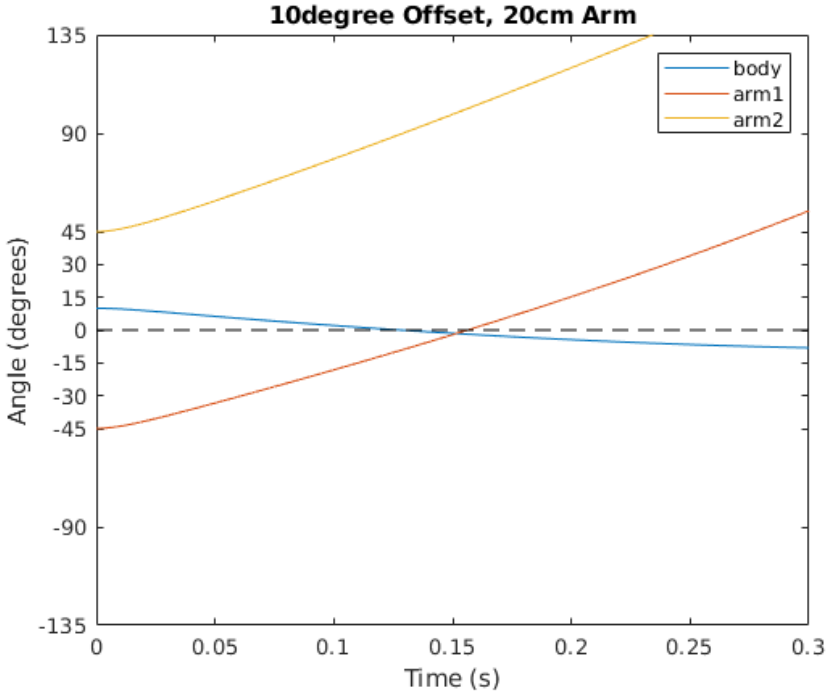


Figure A2. Simulation results of robot recovering from a 10 degree offset. With full torque to the arms, the robot is able to reach 0 degree positioning, within 150 milliseconds.

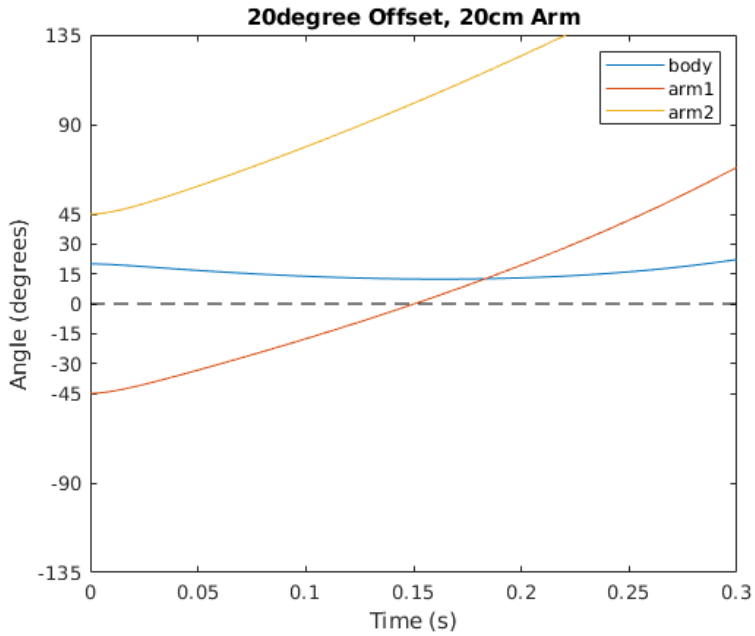


Figure A3a. Simulation results of the robot with 20cm arms recovering from a 20 degree offset. The robot is not close to recovering from this disturbance and can only reach around 10 degrees.

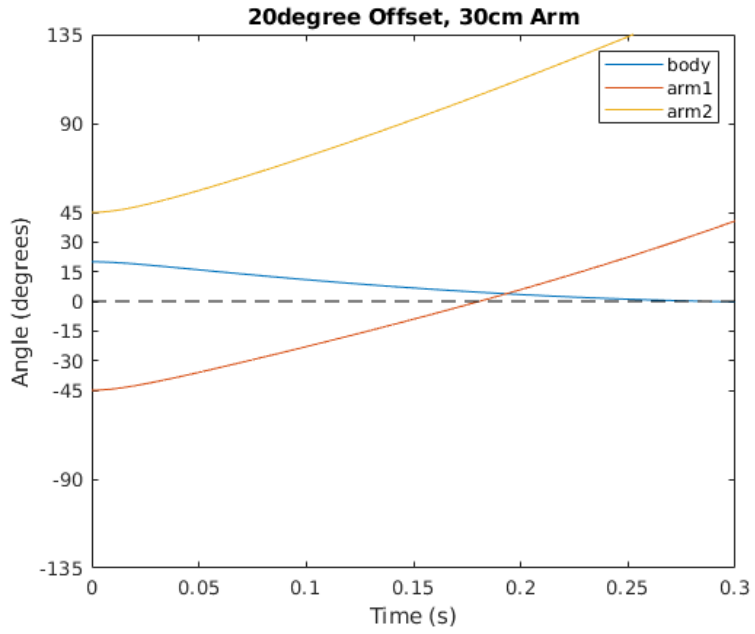


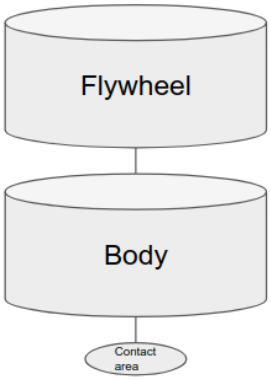
Figure A3b. Simulation results of the robot with 30cm arms recovering from a 20 degree offset. The robot is just able to recover from this disturbance, reaching zero offset angle in 300ms.

The most impactful parameters that can be varied are:

1. Arm length
2. Servo torque/speed
3. COM height of the robot

The arm length is both easy to vary as well as very impactful on the performance of our robot. As shown in the figures above, the robot is close to recovering from a 20 degree offset when the arms are increased to 30 cm whereas the 20 cm arms are capped at around 10 degrees.
It is also important to lower the COM height and mass of the robot as much as possible.

Z-axis rotation



This simulation evaluates the robots ability to overcome friction and rotate on the spot. The simulation models the robot as 2 bodies with angular inertia, one of which is being slowed down by friction. A simple model of the servo is used to determine the torque that can be applied as a function of the rotation rate. The parameters for the model are as follows:

- Flywheel mass: 200g at 7cm
- Body mass: 2.3kg at 9cm
- Motor max speed: 18.8 rad/s
- Motor max torque: 0.08 N*m
- Dynamic coefficient of friction: 0.5
- The static coefficient of friction: 0.9

The results indicate that the maximum possible rotation is very sensitive to the contact area of the wheel against the ground, with larger contact areas leading to smaller total rotation. Another element to note is that the servo torque is a limiting factor, and when the torque due to friction decreases below a certain threshold, the total amount that the robot can turn drops dramatically. If it turns out that the robot is unable to turn on the spot, simulations show that the flywheel can be spun in the opposite direction beforehand, and this increases the amount by which the robot can turn significantly.

Description	Total rotation(radians)
1mm contact radius with pre-rotation	2.450
3mm contact radius with pre-rotation	0.9075
5mm contact radius with pre-rotation	0.3811
1mm contact radius without pre-rotation	0.7850
3mm contact radius without pre-rotation	0.1285
5mm contact radius without pre-rotation	0.0251

Table A1: Summary of simulation results for various contact areas and the associated maximum rotation.

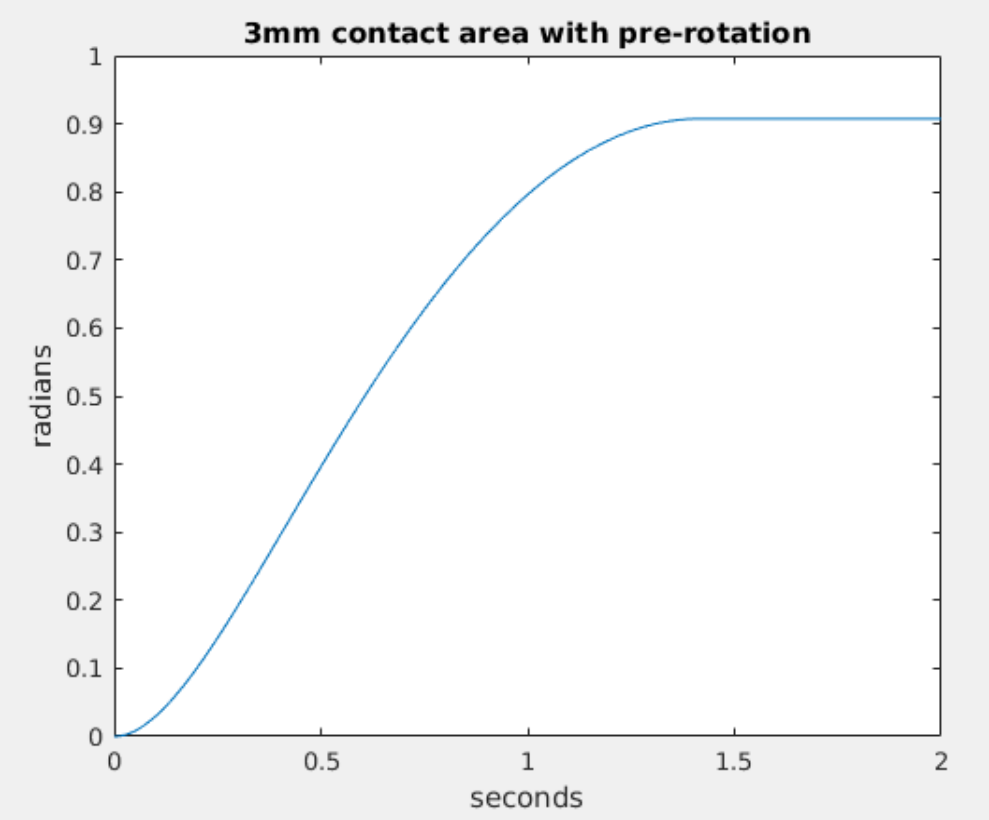


Figure A4. Example simulation of z-axis rotation control, graphing total z rotation in radians against time elapsed in seconds.

B. Motor Characterization

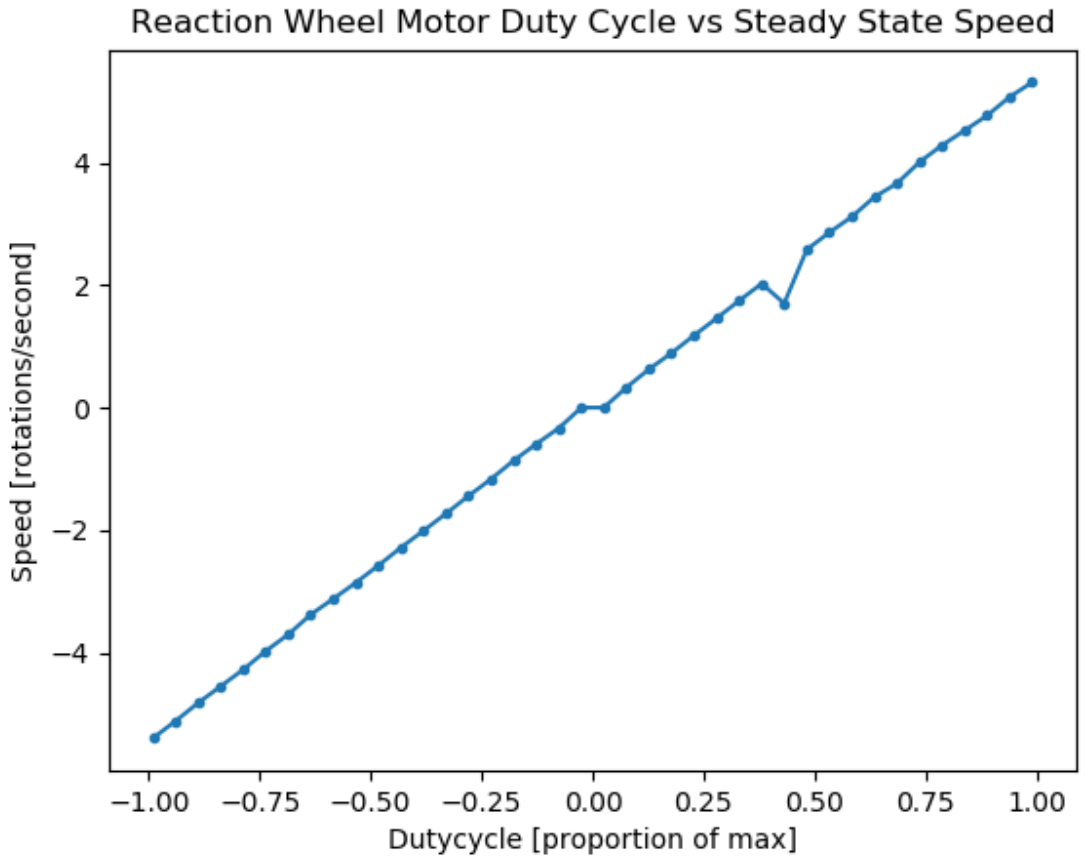


Figure A5. The plot of speed vs PWM signal to Pi. Computing the back-emf constant of the motor by providing PWM signal and recording encoder speed after 5 seconds. The back-emf constant is the slope of the best fit line which is approximately 5.6.

C. Electrical diagrams

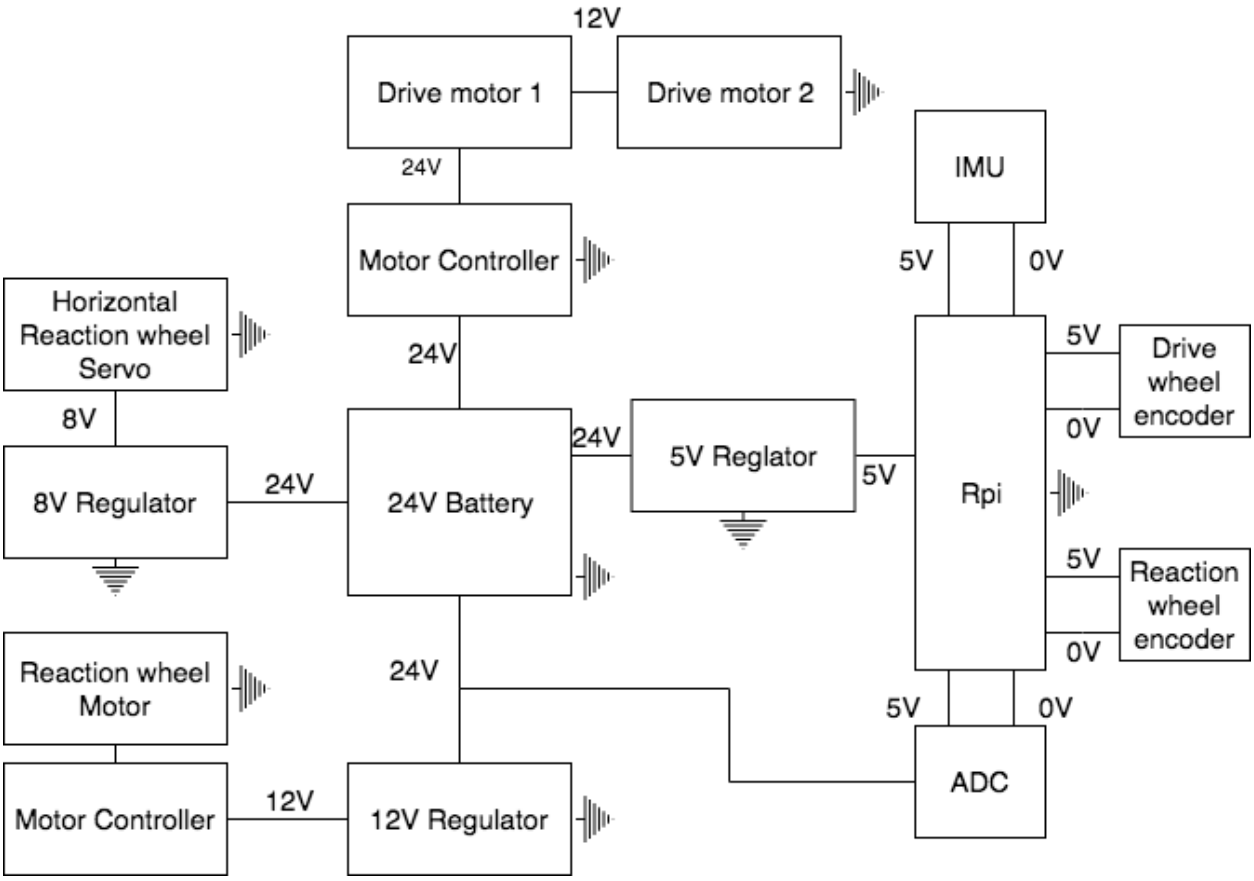


Figure A6. The high-level power distribution diagram for the Unicycle. Power comes from the 24V battery. Power is directly applied to the two drive motors, which are wired in series. The Horizontal reaction wheel servo and the Reaction wheel Motor are driven by 8V and 12V which come from voltage regulators. The Raspberry Pi (Rpi) is powered using a 5V regulator. The IMU, ADC and wheel encoders are all powered from the regulated voltage from the PI.

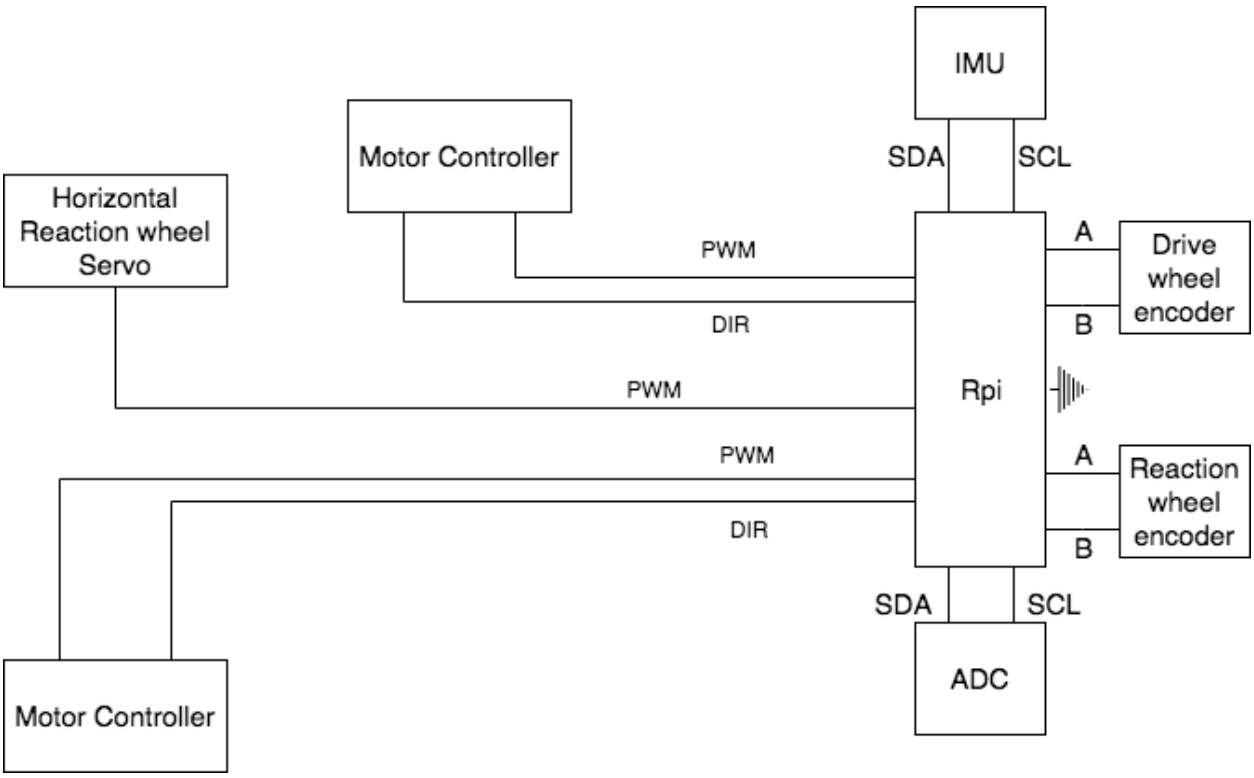


Figure A7. The high-level signal diagram for the Unicycle. Control of the Unicycle is carried out by the Raspberry Pi (Rpi). The Horizontal Reaction wheel receives a single PWM signal that controls the speed of rotation. Both motor controllers for the drive motor and the reaction wheels have a logical direction(DIR) input and a PWM duty cycle input. The ADC and the IMU communicate with the Raspberry Pi using the I2C protocol. The wheel encoders output values along 2 channels, A and B.

D. Details of Computer Vision Tracking

The objective of the overhead webcam tracking system is to determine the 2D position and orientation of the robot. To this end, two differently coloured rectangular markers are symmetrically fixed to the sides of the robot such that their average position roughly equals the center position of the robot. Additionally, the orientation can be determined by the angle formed between the colour markers. We found that using rectangular markers simplified the computer vision algorithm. Initially, circular colour markers were used but they were more susceptible to interference with other objects in the scene.

The marker detection algorithm involves multiple steps:

1. Generate a mask from the image by filtering for pixels within a given colour range covering the colour of the marker as measured by the camera.
2. Find connected areas in the mask and save them as contours.
3. Fit rotated rectangles to the contours and select the contour with the rectangle that best matches the dimensions of the colour markers.
4. Define the center of the tracker to be the average pixel position over all of the pixels of the chosen contour.

E. Usage of the Computer Vision Tracking

The computer vision algorithm relies on OpenCV for python. Version 3.4 of OpenCV was used for all of our tests. Currently, the computer vision tracking script contains a self-contained example (/vision/colour_track.py) as well as an implementation adapted for point-to-point navigation including streaming position and orientation to the robot as well as selecting the destination through the GUI (/vision/point_to_point_demo.py). Both scripts require one command line argument specifying the index of the video stream that should be used by the computer vision tracking.

To be able to use the computer vision tracking algorithm in a new setting, it may be necessary to readjust the camera and colour range parameters due to lighting conditions/height of the camera for example. To reduce the amount of readjustment needed, it is advisable to avoid shadows in the visible range and ensure uniform lighting.

The most important camera settings are saturation and contrast. We set these to 200 and 20 (max is 255) respectively in all of our experiments using the Q4VL2 utility available on most Linux distributions. The video stream for the webcam is typically found at /dev/video/video0.
By default, a black and white video stream is displayed on the GUI with the camera feed and overlaid detections. This black and white video stream corresponds to the colour range filtering. If the colour marker does not appear in this video stream, it is likely that the colour range needs to be readjusted to cover the colour marker better. To do this, one can left click on the colour marker in the colour video field, which prints the associated pixel colour value in the YUV format to STDOUT. One can record these for various different orientations of the robot and then update the colour ranges in the script accordingly. The relevant variables are rightUpper, rightLower,

leftUpper, leftLower. Where left/right denote the side of the robot that the colour marker corresponds to where we define the side-to-side balance reaction wheel as the forwards facing direction. By default, right corresponds to the green marker and the left to the blue. The colours recorded in the previous step should be fully covered by the 3D YUV colour cube spanned between the corner denoted Lower and the corner denoted Upper.

F. Software API

Below we detail our Python API for controlling the robot on the Raspberry Pi. To implement this interface we used the pigpio library. To activate the pigpio library which is required before running any program making use of the interface, one needs to first invoke the command “sudo pigpiod”.

```
class Unicycle:
    def get_imu_data(self):
        """This function returns the data returned by the IMU library. This includes the estimated angular position of the robot as a list of three floats and the gyroscope measurement as a list of three floats as well."""

    def update_battery_voltage(self):
        """Updates the battery voltage used to adjust motor constants based on the ADC reading."""

    def update_counters(self):
        """Updates the wheel encoder counts. This should be invoked at more than 1kHz to accurately measure the wheel rotation."""

    def get_react_motor_encoder(self):
        """Returns the reaction wheel rotation derived from counting the steps in the wheel encoder through update_counters."""

    def get_drive_motor_encoder(self):
        """Equivalent to react wheel motor but for the main drive wheel instead."""
```

```
def write_react_motor(self,value):
    """This function writes a PWM signal to apply torque to the reaction wheel motor proportional to the magnitude of value. The sign of value specifies the direction of the motor rotation. Value should be between -1 and 1."""

def write_drive_motor(self,value):
    """This function is equivalent to write_react_motor but applies to the main drive wheel."""

def write_zrotation_servo(self,value):
    """This function is similar to write_react_motor but applies to the z rotation servo and the value is proportional to the desired speed of the rotation servo rather than torque."""
```

G. Mechanical Design Iterations

Figure A8 below depicts a summary of the mechanical design iterations. The design concept started with the design shown in the figure A8.A1. We then investigated various part configurations and chose the design with the lowest COM, smallest angular inertias, and the highest factor of safety for torque (Figure A8.A4). The mechanical design of the robot was then improved and imported in the pyBullet simulations(Shown in Figure A8.B). Extensive tests in the simulations using different algorithms made apparent that a stable sideways control of the robot using the two arm system requires large masses to be added to the end of the arm and a more complicated controller. In addition, non-repeatability of the servos in combination with the lack of observability of the servo angle in the system further motivated us to move to a simpler design for the sideways control of the robot. The robot shown in Figure A8.C was designed to use a reaction wheel instead of the two arm system to sideway balance the unicycle. This robot was fully made and controlled in the simulation. However, the complexity of the assembly and the number of fasteners pushed us towards a mechanically simpler design. Using the design shown in Figure A8.D we were able to remove all the mechanical fasteners that held the body of the robot and replace them with two plates clamping the components. In this design, we are also capable of moving the COM of the system forward and backward, which allows us to change the mass of the two reaction wheels independently without moving the location of the COM and hence coupling the different axes.

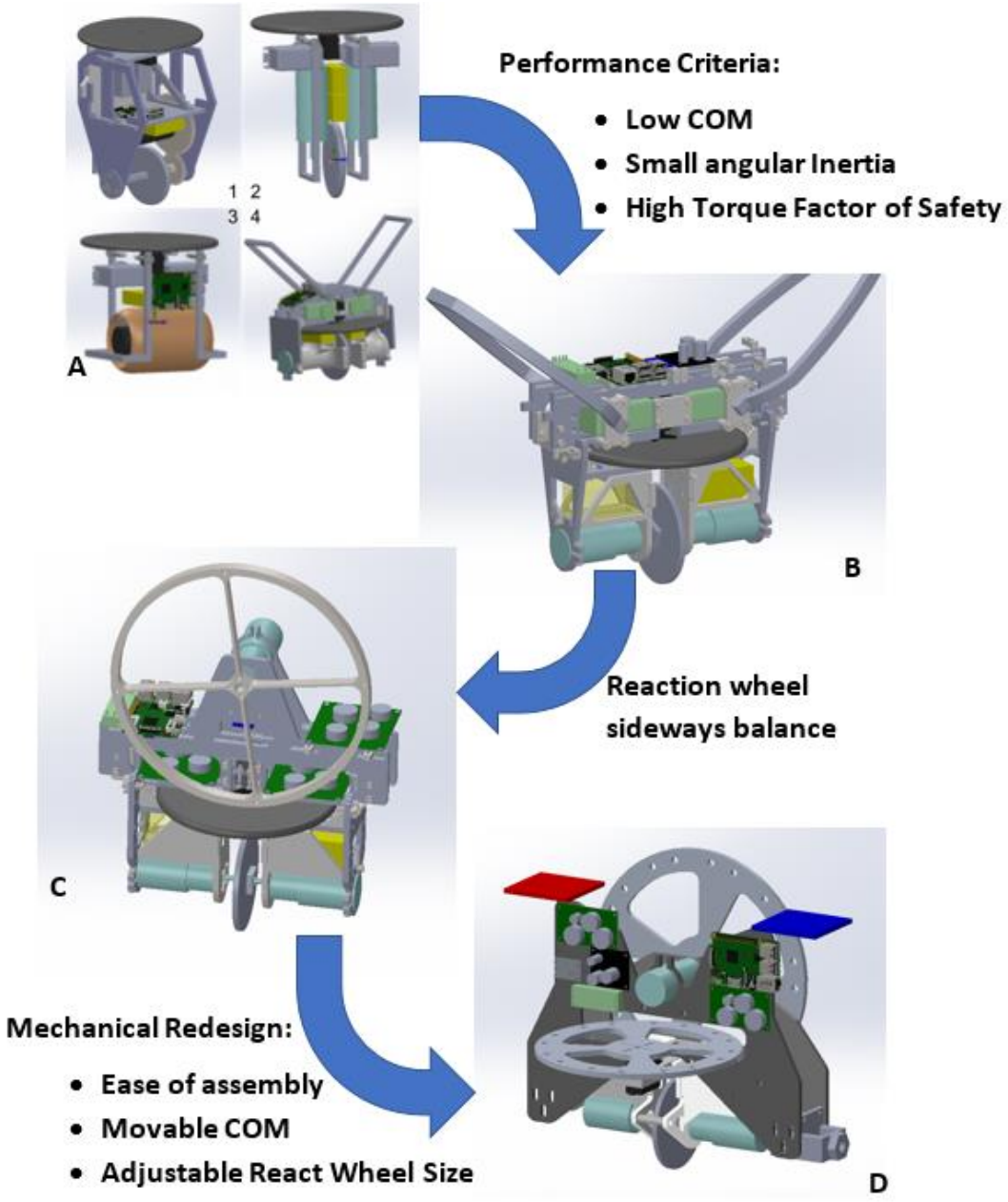


Figure A8. A Summary of the Mechanical Design Iteration. Multiple configurations (shown in figure A8.A) were investigated. According to the performance criteria, the details of the most optimum design was completed(Figure A8.B). Controller complexity pushed us to use a reaction wheel design for the sideways balance of the robot(shown in Figure A8.C). The robot was redesign to allow for ease of assembly, a movable COM, and the ability to change the reaction wheel size and mass(Figure A8.D)

H. Intermediate Test Design and Results

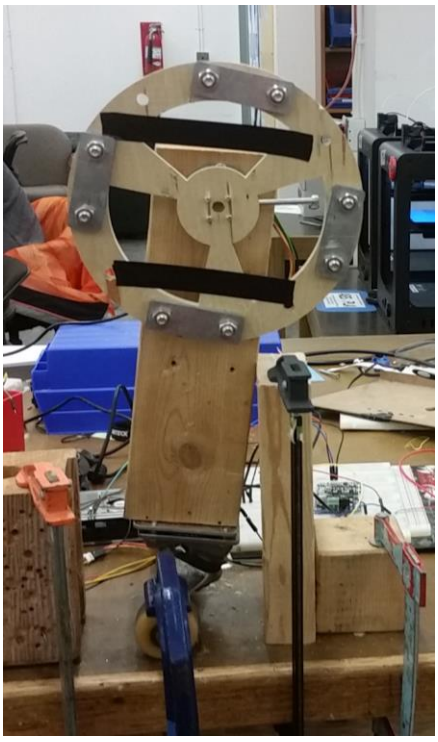


Figure A9. A picture of the reaction wheel test rig used during the intermediate tests to validate the IMU readings and motor capabilities.

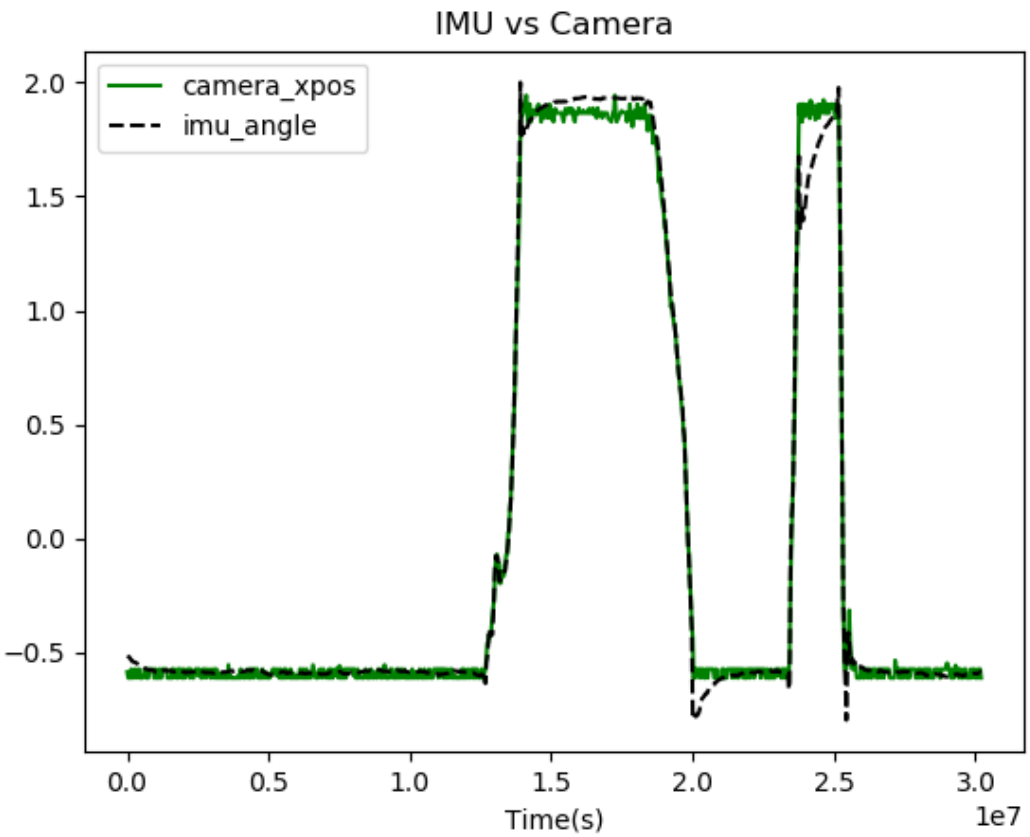


Figure A10. The test rig as seen in Figure A9 was equipped with a color marker and the color tracking script was used to record the position of the marker. For small angles, the x-position is proportional to the angle of the robot and is plotted against the rotation as measured simultaneously by the IMU. Both signals were normalized to bring them to an equal scale. The test rig is actuated by the motor and the resulting sensor readings are plotted. As can be seen in the figure, the IMU sensor is very responsive and tracks the computer vision tracker very well except for the hard collisions that can be observed at the ends of the step function in the camera signal.

References

[1] Li, & Yuxi. (2018, November 26).
Deep Reinforcement Learning: An Overview.
Retrieved from https://arxiv.org/abs/1701.07274

[2] Xie, Berseth, Glen, Clary, Patrick, Hurst, . . . Michiel. (2018, July 27).
Feedback Control For Cassie With Deep Reinforcement Learning.
Retrieved from https://arxiv.org/abs/1803.05580

[3] Peng, Bin, X., Andrychowicz, Marcin, Zaremba, Wojciech, . . . Pieter. (2018, March 03).
Sim-to-Real Transfer of Robotic Control with Dynamics Randomization.
Retrieved from https://arxiv.org/abs/1710.06537

[4] Lee, J., Han, S., & Lee, J. (2013).
Decoupled Dynamic Control for Pitch and Roll Axes of the Unicycle Robot.
IEEE Transactions on Industrial Electronics, 60(9), 3814-3822. doi:10.1109/tie.2012.2208431

[5] Saltyzombie. (2011, June 09). Unicycle Countersteer. Retrieved from
https://www.youtube.com/watch?v=55n90QAxRSk

[6] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., ... & Zaremba, W. (2017).
Hindsight experience replay. In *Advances in Neural Information Processing Systems*(pp. 5048-5058).

[7] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., & Hadsell, R. (2016). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.